A Systematic Approach to Evaluating Open Source Software

Norita Ahmad
School of Management and Business,
American University of Sharjah,
P.O. Box 26666, Sharjah UAE
nahmad@aus.edu

*Phillip A. Laplante
Engineering Division,
Penn State Great Valley School of Graduate Professional Studies
30 East Swedesford Rd, Malvern, PA 19341, USA.
plaplante@psu.edu

Abstract

Selecting appropriate Open Source Software (OSS) for a given problem or a set of requirements can be very challenging. Some of the difficulties are due to the fact that there is not a generally accepted set of criteria to use in evaluation, and that there are usually many OSS projects available to solve a particular problem. In this study, we propose a set of criteria and a methodology for assessing candidate OSS for fitness of purpose using both functional and non-functional factors. We then use these criteria in an improved solution to the decision problem using the well-developed Analytical Hierarchy Process. In order to validate the proposed model, we applied it at a technology management company in the United Arab Emirates, which integrates many OSS solutions into its Information Technology infrastructure. The contribution of this work is to help decision makers to better identify an appropriate OSS solution using a systematic approach without the need for intensive performance testing.

Keyword

Open Source Software, Analytical Hierarchy Process, Evaluation Process, Evaluation Criteria, Systematic Approach, Decision Making, Free Open Source Software, Free/Libre Open Source Software

Introduction

The terms “Open Source Software” (OSS), “Free Software”, "Free Open Source Software" (FOSS), and "Free/Libre Open Source Software" (FLOSS) are often treated synonymously (Feller and Fitzgerald, 2002; Feller, et al., 2005; Koch 2005). When we look at their respective license agreements, however, we can easily see that they are quite different. Free software is generally licensed with the GNU General Public License (GPL), while OSS may use either the GPL or some other license that allows for the integration of software that may not be free (Elliott and Scacchi, 2008; Gay, 2002). Free software is always available as OSS, but OSS is not always free software. Therefore it is more appropriate to refer to FOSS or FLOSS instead of the more general term "open source" in order to differentiate between the two different models and preserve the original meaning of the free software/FOSS/FLOSS. We would like to note that in this paper, when appropriate, we used terms specific to either free software or OSS when such differentiation is necessary.

Since most OSS is free to use and modify with no licensing fees, it is attractive for use by many including government, businesses, and non-profits (Feller, Fitzgerald, Hissam, and Lakhani, 2005). However, it can be difficult to evaluate or choose the right OSS. One of the unique challenges of evaluating OSS is the sheer number of OSS projects available (Feller, and Fitzgerald, 2002). Anyone can create an OSS project on a free hosting site such as SourceForge.net. This low barrier to entry means that many OSS software projects are very immature (Deprez and Alexandre, 2008; Gacek, and Arief, 2004). Another challenge is that OSS projects often have little documentation (Wheeler, 2005). Without proper documentation and user manuals that traditionally accompany commercial software, it can be difficult to confirm an OSS' feature set.

Balancing the challenges of evaluating OSS software are the unique advantages provided. The biggest advantage is that the source code is available for analysis, which is vital in determining whether the software is of high quality and is maintainable. Another advantage is that many OSS projects provide public read-only access to their issue tracking system, which can give valuable insight into how fast the project is growing, whether defects are being found and fixed, and the amount of time it takes to resolve issues.

Selecting appropriate OSS for a given problem or a set of requirements can be very challenging. Some of the difficulties are due to the fact that there is not a generally accepted set of criteria to use in evaluation, and that there are usually many OSS projects available to solve a particular problem. Therefore the evaluation is often done in an ad hoc manner, using whatever criteria were available to the evaluators (Conradi, Bunse, Torchiano, Slyngstad, and Morisio, 2009; Norris, 2004). This kind of approach leads to evaluations that are not systematic or standardized within or between organizations, and are not repeatable, which in turn, could slow down project development. Another known problem is that, the evaluation process often lack operational approach where not everybody is involved in the evaluation process (Merilinna and Matinlassi, 2006; Torchiano and Morisio, 2004).

In this study, we propose a rigorous selection methodology for assessing candidate OSS using both functional and non-functional factors based on a set of criteria (Confino and Laplante, 2010) where every stakeholder in an organization can be involved. In order to test this evaluation model several important OSS were examined. We present an improved solution to the problem using the well-developed Analytical Hierarchy Process (AHP), which is not traditionally used or advocated by software developers/engineers. We also surveyed fifteen experts from a technology management company in the United Arab Emirates (UAE) which integrate many OSS solutions into its infrastructure. The contribution of this work is to help the decision maker to make a better decision in identifying an appropriate OSS solution using a systematic and repeatable approach without the need for intensive performance testing.

Previous and Related Work

An Open Source Software (OSS) program is a piece of software, with its source code available, that any person can access, use and copy provided that the associated license provisions are honored. A more comprehensive and formal definition of OSS can be found at the Open Source Initiative web site (Open Source Initiative, 2010). Many have argued that the definition provided by the Open Source Initiative is not sufficient and therefore devised a few more explanations (Feller and Fitzgerald 2002; Lawrie, Arief and Gacek 2002; Nakakoji

et al. 2002; Raymond 2000; Sharma, Sugumaran & Rajagopalan 2002; So, Thomas and Zadeh 2002).

The open source development process is significantly different from the traditional commercial off-the-shelf (COTS) model. The most well-known model is Raymond's 'Cathedral and the Bazaar' metaphor (Raymond, 2000). Raymond compared conventional software engineering to a cathedral, that is, a highly formalized, well-defined and rigorously-followed development processes. On the other hand, he characterized the OSS development approach as a bazaar style of development where everyone is free to develop in their own way and to follow their own style.

Open Source Software Phenomenon

The term 'Open Source' was first introduced in 1998 (Open Source Initiative, 2000). The historical context of the movement includes the history of Unix operating system (Hauben & Hauben 1997; Salus 1995), the Internet (Hauben & Hauben 1997; Licklider & Taylor 1968), and the hacker culture (Levy 1984; Raymond 2000c; Turkle 1984). The Free Software Foundation and the GNU project also played a very significant role in this movement (Feller & Fitzgerald 2002; Levy 1984; Moody 2001). Open source became more popular when businesses started to show interest in it (Apple Computer Inc. 2002; Hamerly, Paquin & Walton 1999; IBM 2003; SGI 2003; Sun Microsystems Inc.). One such phenomenon is the case of how Linux was employed as a weapon against Microsoft and other competitors (Bezroukov 2002; Wladawsky-Berger 2001).

OSS has attracted considerable attention since it appears to address the three main aspects of software development: cost, time-scale and quality (Feller, and Fitzgerald, 2002). Since OSS is freely available for public download the cost is not an issue at all. In terms of development speed, OSS has an advantage as well given the collaborative nature of its community where developers are globally distributed. OSS also has a good reputation in terms of quality. Many OSS products such as Apache and Linux are known for their reliability and robustness (Mockus, Fielding & Herbsleb, 2002; Raymond, 2001).

In addition, an OSS communities also includes virtual communities and virtual organizations (Crowston & Scozzi 2002; Dafermos 2001; Gallivan 2001; Kollock 1996; Markus, Manville & Agres 2000; Rheingold 1993; Romm, Pliskin & Clarke 1997; Sharma, Sugumaran & Rajagopalan 2002; So, Thomas & Zadeh 2002; Wellman & Gulia 1999), the current state of hacker culture (Moody 2001; Pavlicek 2000; Raymond 2000b, 2000a), information economy (Clarke 1999; Ghosh 1998a; Kollock 1999; Lancashire 2001; Lerner & Triole 2002) and the political influences of Open Source (Forge 2000; Free Software Foundation 2002; Newman 1999; The Associated Press 2000; Yee 1999). The communities are mostly made up of members with technical backgrounds (Lakhani et al. 2003) and therefore technology plays a very important role in the OSS community. Topics related to architecture such as the microkernel vs. monolithic debate (DiBona, Ockman & Stone 1999), and features, such as technical supremacy of Linux over Microsoft (The Unix vs. NT Organisation, 2001) of software were always important focuses in the communities.

Open Source Software Evaluation Models

There are at least four open source software evaluation models available today. The first one, The Open Source Maturity Model (OSMM), was created in 2003 by the Capgemini

consulting company. It employs 12 weighted criteria, which can be input into a spreadsheet for evaluation (Duijnhouwer and Widdows, 2003). The second model, also called The Open Source Maturity Model (OSMM), was created by Navica in 2004. Navica's OSMM seeks to assess the maturity of six key elements of software, and uses spreadsheets to assist in the evaluation (Golden, 2004). The third model, The Qualification and Selection of Open Source software (QSOS), was created in 2004 and sponsored by the consulting company Atos Origin. The QSOS also uses a set of evaluation criteria, and provides web-based tools to assist in the evaluation process. The website also contains a database of previous evaluations (QSOS, 2009). The forth model, The Business Readiness Rating (BRR), was created in 2005 and sponsored by Carnegie Mellon West, SpikeSource, O'Reilly, and Intel. The BRR provides evaluation criteria and spreadsheets for evaluation as well (OpenBRR, 2009). Interested reader can refer to (Deprez, 2009) for a comparison of QSOS and BRR and (Wikipedia, 2009) for a comprehensive review and comparison of all the four models.

In addition to the four formal evaluation models described above, there are other important related studies by Confino and Laplante (2010), Wheeler (2005), Crowston et. al. (2004), and Donham, (2004).

Model Development

The criteria for the evaluation model proposed in this study were constructed using the Open Source literature on the subject of evaluation of software. As discussed in Section 2.2, there are at least four formal open source software evaluation models and several other sets of criteria for other researchers. From these criteria sets we identified eight common criteria: Functionality, Licensing, Product Evolution/Velocity, Longevity/Pedigree, Community, Market Penetration, Documentation Quality, and Support. From our analysis, we were able to identify a common weakness in all of these models as well, that is, they are too high level. In addition, only one of these models (Confino and Laplante, 2010) included an evaluation of the actual source code criterion. Since code quality evaluation is an important part of determining the overall fitness of a piece of software, we include a source code quality criterion, leading us to a total of nine evaluation criteria. Table 1 lists the nine main criteria used in this study and how they were employed in each source mentioned in the previous section.

It is important to note that in creating a feature list for evaluation in this study, both functional and non-functional features are included. Functional features describe the required behavior of the software (Colombo and Francalanci, 2004). Nonfunctional features are the properties or characteristics that the software must have such as usability, quality or reliability. In this study we have identified two functional features and seven non-functional features (see Table 2).

Table 1. Literature on OSS evaluation criteria

Criteria	OSMM CapGemini	OSMM Navica	QSOS	BRR	Crowston et al. 2004	Wheeler 2005	Donham 2004	Confino & Laplante
Functionality	Compatibility	Integration	√	√	-	√	√	√
Product Evolution	-	Activity Level	Activity	-	Activity Level	Maintenance	-	Velocity
Licensing	√	In Risk	√	-	-	√	√	√
Longevity/Pedigree	-	√	Maturity			√	Maturity	√
Community	√	√	√	√	Team Size	Combined with support	-	√
Market Penetration	√	-	Popularity		-	√	-	√
Documentation	-	√	√	√		Combined with support	√	√
Support	√	√	√	√		√	√	√
Code Quality	-	-	Maintainability	-	-	-	-	√

Note: A criterion is listed "√" if it is mentioned in the article, and "-" if it is not listed. If it is listed under a different term then we use the term (i.e. 'Compatibility) or if it is part of another section then we mention the section name (i.e. 'In risk').

Table 2. A summary of the evaluation criteria for open source software

	Criterion	Descriptions
Functional Features		
1	Functionality (m_1)	Used to indicate whether the software possesses the required features. By using the same standard to evaluate multiple products some objective comparisons can be drawn.
2	Product Evolution (m_2)	Used to indicate if the community has developed clear thoughts and plans about which features will be changed or added in the future.
Non-functional Features		
3	Licensing (m_3)	Used to indicate the appropriateness of the software's license for the intended usage.
4	Longevity/Pedigree (m_4)	Used as a quality measure of the authors, patrons, and lineage of the project. Organization and individuals who have produced quality software in the past are more likely to produce quality software.
5	Community (m_5)	Used as a quality measure of the community participating in an open source project.
6	Market Penetration (m_6)	Used to indicate the market penetration of the source software.
7	Documentation Quality (m_7)	Used as a measure of the quality of the documentation
8	Support (m_8)	Used as a measure of the quality of the support, both commercial and community, available for the software.
9	Code Quality (m_9)	Used indicate the quality of the source code.

Overview of the evaluation criteria

In this section, each of the criteria listed in Table 1 is described in detail, giving background information and explaining why the criterion is important for the evaluation. No ranges are given for these criteria because they will be evaluated based on relative importance.

Functionality (m_1)

The functionality criterion, m_1, is used to indicate whether the open source software possesses the required features. Functionality is often used in most traditional software evaluation models and is not specific to OSS evaluation. Functionality is an important criterion because in order for software to be useful it should have all the features needed to meet project requirements. Open Source projects might not list the complete information on functionalities on the website (Golden, 2005) but the software can be installed for discovery of the functionality needed.

Project Evolution/Velocity (m_2)

Project evolution, m_2, is used indicate if the OSS community has developed clear plans about features that will be added or improved. It can also be used to evaluate if the candidate OSS is a healthy, growing project, or a stagnant, dying project. Many open source projects grow stagnant after the initial author(s) depart(s). A healthy project usually has regular software releases with added or improved functionality and a fairly constant pace of development activities. There are multiple sources for gathering data about an open source project's releases: a download page on the project's website, a change log, or the source code repository.

Licensing (m_3)

The licensing criteria, m_3, is used to indicate the appropriateness of the open source software's license. It is very important to understand an OSS licensing terms because the licenses vary so greatly in terms of permissiveness of use and redistribution and the negative implications of misusing the license are profound. Currently, there are more than one hundred different open licenses (OSS Licenses, 2010) Organized into two general categories, permissive licenses, and copyleft licenses. Copyleft licenses can then we divided into strong copyleft licenses, and weak copyleft licenses. Interested readers can refer to (OSS Licenses, 2010) for detail discussion of these licensing categories.

Longevity/Pedigree (m_4)

The software pedigree criterion, m_4, is use to evaluate the authors, patrons, and lineage of the project. Pedigree gives an indication of what is known of the community, culture, and processes that created the software. It is often assumed that organization and individuals who have produced quality software in the past are more likely to produce quality software now. In addition, it is also assumed that software derived from other high quality software is likely to be of high quality.

Community (m_5)

The community criterion, m_5, considers the candidate software's community as an indirect measure of the software project's overall health. OS communities consist of the people that

use the software, test the software, and provide quality feedback or participate in some way. A large, diverse and vibrant community is usually a good sign of project health and sustainability. Other signs of a good community include an updated website, an active mailing list, and an active message board or Wiki where information between community members is being exchanged. The more people are interested in a project, the more likely it is that the project will be active and keep growing. A large and active community says something about the acceptance of the software. If the software was not good enough for the public, then not many people would be interested in participating in its development (Duijnhouwer and Widdows, 2003).

Market Penetration (m_6)

Criterion m_6 is used to evaluate an OSS's market penetration. Determining a market leader of OSS is a subjective endeavor. Objective metrics such as sales figures or installed client base, which are often used in other industries, are not available for OSS since users can simply download software anonymously. However, there are several sources that can be used to indirectly measure OSS market penetration. These sources include a number of reviews in trade magazines, websites, and blogs. For example, sites such as Ohloh (http://www.ohloh.net) provide community reviews, Google Trends (http://trends.google.com) can be used to gauge popularity of the project's website and for Linux-based software, inclusion in highly regarded Linux distributions such as Red Hat can be a sign of market penetration (Wheeler, 2009).

Documentation Quality (m_7)

Criterion m_7 is used to assess the quality of the open source software's documentation. Even though documentation is a critical component to software fitness, many open source software projects have poor documentation (Wheeler, 2009). Still, it is important to understand and thoroughly evaluate whatever documentation is available. In OSS project, documentation can sometimes be found at the project site or in related sites. Other documentation-like artifacts can be found in code comments, separate documentation files, help features within the application itself, project and user forums, wikis, and message boards.

Support (m_8)

Criterion m_8 refers to the quality of the support available for the open source software under evaluation, including support provided by the community and by third parties. Community support is freely provided by the software's developers and by the user community. Support services are usually found on the project's website, mailing lists, defect tracking systems, and message boards. Third-party, commercial support is often provided by the author of the software or a company specializing in support for OSS. This third-party support usually includes technical support, bug fixes, frequently asked questions lists, and indemnification based on some fee or subscription model.

Code Quality (m_9)

The code quality, m_9, is used to evaluate the quality of OSS code. Evaluating the source code gives users key insights into the quality of the product that are not observable otherwise. There are three main types of code quality analysis, static, dynamic, and design. Static analysis looks at the static structure of the code. Metrics such as cyclomatic complexity,

cyclic dependency count, and number of lines per method are usually used to determine whether code is well structured. While the metrics themselves are completely objective, the analysis of the metrics is subjective. Dynamic analysis is usually conducted during a performance test where memory allocation, CPU consumption, and I/O usage are measured during program execution.

Design analysis is a subjective evaluation of the code structure at the design or architectural level, for example, if the code is optimally designed using object-oriented design principles (Martin, 1999). Design analysis is usually quite difficult, time-consuming, and often arbitrary, even using open source or commercial tools. Instead, we suggest that it is more effective to focus the design evaluation activity to a few key portions of the code.

Framework Development Using Analytical Hierarchy Process

In order to systematically evaluate the most suitable OSS software for a given project, we apply a novel application of a traditional technique for multivariate decision making called Analytical Hierarchy Process (AHP). In particular, we formalize the problem statement "how does one select appropriate OSS for a specific application?"

We chose AHP because it is ideal for complex, multi-criteria problems where both qualitative and quantitative aspects of a problem can be incorporated (Ahmad, 2005). In AHP, any given problem is structured in terms of a hierarchy (see Figure 1).

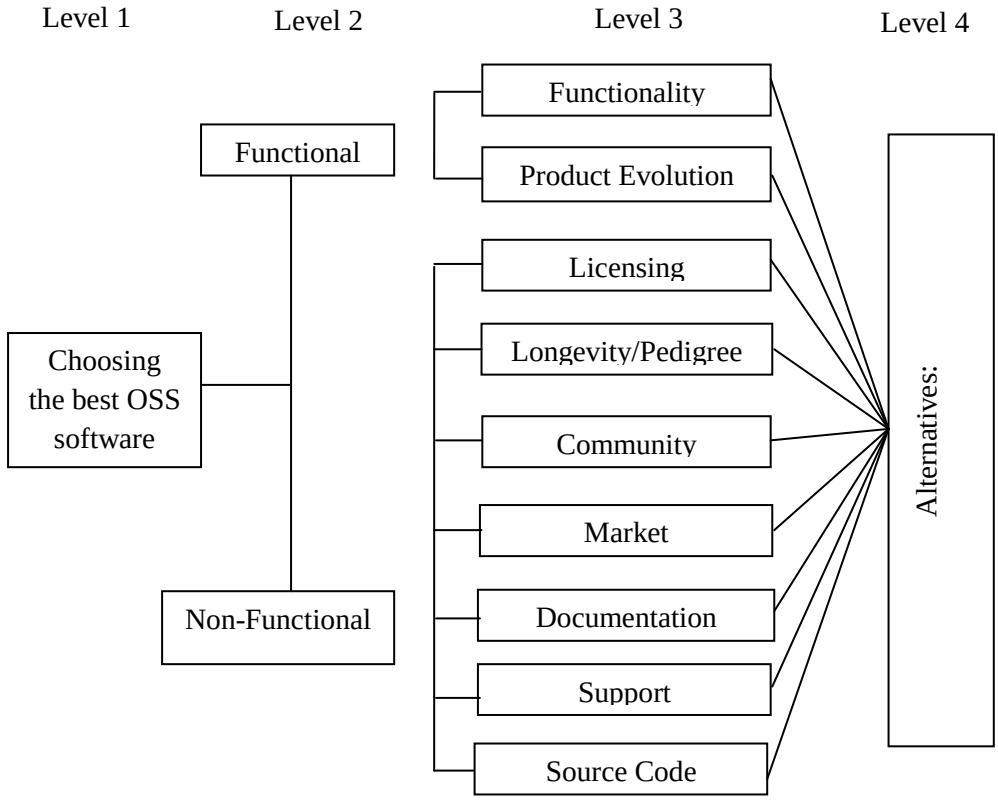

Figure 1: Hierarchy structure of choosing OSS software (4 level hierarchy: Objective, Criteria, Sub-criteria and Alternatives

Note: *Each alternative in level 4 is connected to every criteria in level 3*

AHP simplifies the decision-making process by breaking a complex problem into a series of structured steps as follows.

- Define the objective.

The first step in the AHP process is to define the objective. In our case the objective is "Choosing the best OSS software."

- Identify the criteria that influence the behavior.

Once the objective is defined, we will then define the criteria to support the objective, followed by any sub-criteria or sub sub-criteria depending on the complexity of the problem. In our case, we have two main criteria and nine sub-criteria as shown in Table 3.

- Identify the alternatives

After all the sub-criteria or sub sub-criteria are defined, we can then look at the alternatives that we want to measure in the problem. The alternative is always placed at the lowest level of the hierarchy. In our case, we evaluated 5 different OSS applications in the "collaboration/groupware/communication" category (see Figure 1).

- Perform pair-wise comparisons

Once the hierarchy is constructed, we then compare each element in the corresponding level and standardize these on a numerical scale. This step requires n(n-1)/2 comparisons, where n is the number of elements. A ratio of relative importance is assigned to each paired comparison, usually according to the Saaty linear nine-point scale*, {1/9, 1/8, 1/7, 1/6, 1/5/, 1/ 4, 1/3, 1/2, 1, 2, 3, 4, 5, 6, 7, 8, 9} w*here the weights have the interpretations given in Table 3. Comparisons of a criterion with itself are always unity and if criterion A is rated as having a relative importance of R to criterion B, then the relative importance of criterion B to criterion A must be 1/R.

- Perform calculations

Once the comparison matrix is constructed, we would then find the maximum Eigenvalue, consistency index (CI), consistency ratio (CR), and normalized values for each criteria/alternative. If the maximum Eigen value, CI, and CR are satisfactory then decision is taken based on the normalized values, otherwise we would repeat the procedure until these values are within the desired range (Saaty, 1980).

Table 3. Weighting factors used in AHP.

Weight	Interpretation
1	Equally Preferred,
2	Equally to Moderately Preferred,
3	Moderately Preferred,
4	Moderately to Strongly Preferred,
5	Strongly Preferred,
6	Strongly to Very strongly Preferred,
7	Very strongly Preferred,
8	Very to Extremely Strongly Preferred,
9	Extremely Preferred

The hierarchy method used in AHP has various advantages. One of the most prominent is the ability to incorporate a group decisions. Generally each members of the group performs his own pair-wise comparisons and combines only their final outcomes obtained in his hierarchy by taking the geometric mean of the final outcomes of the other hierarchies. In the case where

the individuals have different priorities of importance, their judgments are raised to the power of their priorities and then the geometric mean is formed (Saaty, 1980). This approach is a powerful way to build consensus, as each member can see where they stand and compare their judgments to those of the group.

Methodology

In order to test the framework defined in Figure 1, we interviewed and surveyed fifteen experts from an Information Technology Management Company (ITM). ITM was established in 1994 in Dubai, UAE and currently has about 50 full time employees. This company integrates many OSS solutions into its Information Technology (IT) infrastructure. Recently the company deployed a new groupware application and again it opted for an OSS application. Typically, any evaluations and decisions related to software and application adoption are conducted by the company's IT department. No formal evaluation criteria or tools were used before. While software availability seems to be the most common criteria for ITM, selections were usually based on functionalities required, the experiences of the developers, recommendations made by vendors, and popularity of the software/application.

This situation provided an excellent opportunity to test our evaluation method. At the time of first approaching the company, ITM had already identified five candidate collaboration/groupware applications as shown in Table 4.

Table 4. OSS Candidates

Application	**Project Website**
Zimbra 6.0	http://www.zimbra.com/
Scalix 4.2,	http://www.scalix.com/
Open-Xchange 6.12	http://www.open-xchange.com/oxpedia
eGroupware 1.6	http://www.egroupware.org/
Kolab 2.2.0	http://www.kolab.org/index.html

The list was compiled by the IT director after conducting Internet searches. Typically the IT director would work very closely with his staff from the IT department in selecting the application. In order to effectively test our method we asked the IT director to include people from other departments to participate in the evaluation process. Participants selected ranged from the most basic users to the most technical users – a fair representation of all stakeholder groups in ITM. In addition, all participants selected had prior experience with the OSS application, and hence, they were representative of OSS users.

We had two meetings with all participants during this study. The first meeting was held in the beginning of the evaluation process where we explained the study and the evaluation process in detail. The IT director also made all the OSS candidates' software (demo version) available to every participant so that they could test functionalities of each application. The second meeting was held at the end of the study where we discussed the result with everyone and obtained their feedback about the entire evaluation process.

Survey Sample

The participants (experts) for this study are five developers, five end users and five upper level managers (who are also the end-users) in the Information Technology Management Company in the United Arab Emirates (UAE). All participants were promised anonymity and

confidentiality of their participations; therefore we will refer to the participants as DExpert1-5 for developers, EExpert1-5 for end-users and MExpert1-5 for the upper level managers.

The participants were selected based on their job descriptions and their experience in the OSS environment. All of these participants are the major players in their respective departments. In addition, the selection covered different categories of users that are technical and non-technical. The survey questionnaires were e-mailed to the fifteen experts who had agreed to participate in judgmental exercises involved in the AHP. The experts were given two to three weeks to complete the survey. By the deadline, all fifteen experts (30% of population) have successfully completed and returned the survey. It is important to note that the results obtained from this convenience sample of subjects represent a broad cross section of experts' opinions (see Table 5) and can guide us about decision making and perception towards OSS.

Table 5. Sample demographics

Experts	Title	Education Background	Years of OSS Experience
DExpert1	Software Developer	Bachelors	3-5 years
DExpert2	Database Administrator	Bachelors	3-5 years
DExpert3	Senior Software Developer	Bachelors	6-10 years
DExpert4	Senior Systems Architect	Bachelors	3-5 years
DExpert5	Systems Design Specialist	Bachelors	6-10 years
MExpert6	Director of Information Technology	Masters	6-10 years
MExpert7	Vice President of Operations	Bachelors	3-5 years
MExpert8	Director of Finance	Masters	3-5 years
MExpert9	Director of Consulting and Support Services	Masters	3-5 years
MExpert10	Vice President of Sales and Marketing	Masters	3-5 years
EExpert11	Accountant	Bachelors	3-5 years
EExpert12	Customer Support Specialist	Bachelors	3-5 years
EExpert13	Associate Consultant	Bachelors	≤ 2 years
EExpert14	Administrative Support	Certificate	≤ 2 years
EExpert15	Senior Business Systems Analyst	Masters	3-5 years

Description of the survey procedures

The survey was designed by encapsulating the AHP and allowing users to develop their own preferences using the basic factors supplied by the survey. The use of AHP allows the collection of ratio data rather than ordinal data, which carries only the rank order of the related data items. Ratio scales are necessary to represent proportion or the value of the relative importance of each factor indicated by users. The survey data are structured into a hierarchy that includes all the common criteria that give a good general idea of the OSS software that needs to be evaluated.

Based on the hierarchy described in Figure 1, the experts then made judgments on the elements in the hierarchy on a pairwise basis with respect to their parent element e.g. in identifying the most suitable OSS Software, is functional factor more important than non-functional factor and if so, by how much? Similar comparisons are performed at each level in the hierarchy.

In order to ensure the accuracy of pairwise comparisons, during our first meeting all experts were given an instruction on how to perform the comparison using our AHP software. In addition, detailed instructions and examples were given on the survey itself on how to conduct the comparison among the elements (alternatives) with respect to the immediately preceding criterion on the hierarchy. A bottom-up approach is used where the alternatives (Level 4) are first compared with respect to each sub-criteria (Level 3). Next, a comparison is made where the sub-criteria (Level 3) is compared with respect to the criteria (Level 2). Finally, the criteria at Level 2 are compared among themselves with respect to the objective (Level 1). By starting at the lowest level, the experts gain familiarity with the details of the higher level decision attributes before making those higher level paired comparisons.

The survey questionnaires were e-mailed to the fifteen experts who had agreed to participate in judgmental exercises involved in the AHP. In order to ensure that the result of the evaluation is reliable and the process is carried out fairly, the experts were given access to the demo version of each OSS application and link to the projects' websites. The experts were given two to three weeks to complete the survey. All fifteen experts had successfully completed and returned the survey by the deadline.

When performing the pairwise comparison users have to evaluate criteria one pair at a time, given n number of criteria, there are precisely n(n-1)/2 criteria to be evaluated. There are many software programs available to help with the calculations, for example, Expert Choice (http://www.expertchoice.com/). In this study, we used Java-based AHP software that was developed by one of the authors to perform all the calculations.

Since the experts used the AHP software provided by the authors for pairwise comparisons, any inconsistency in their judgments' will be reported real time. In order to achieve consistency and to better reflect their perception and understanding of the evaluation, the experts were allowed to revise their comparisons. The experts were also free to discuss their responses with each other and revise their input until they are completely satisfied with the outcomes. Therefore, the results of their final judgments were the results of many revisions that the experts have gone through. Once we received the results from all the experts, we then combined them into a single representative judgment for the entire group.

Data Analysis

The fifteen experts evaluated the hierarchy of the nine variables (Figure 1) constructed by pair-wise comparison. Since the model consists of more than one level (Level 1 is the objective, Level 2 is the main criteria, Level 3 is the sub criteria, and Level 4 is the alternative), hierarchical composition was used to weigh the eigenvectors by the weights of the criteria. The sum was taken over all weighted eigenvector entries corresponding to those in the lower level, and so on, resulting in a global priority vector for the lowest level of the hierarchy. The global priorities are essentially the result of distributing the weights of the hierarchy from one level to the next level below it.

As an example, consider the hierarchy in Figure 1. Assume that Functional factor has been given a comparative weighting of 0.4 by AHP and Non-functional factor a weight of 0.6 (at each level, the local weights will sum to 1). When assessing the importance of the Functional factors, assume that AHP computes a weight of 0.8 for Functionality (m_1) and 0.2 for Product Evolution (m_2). When these weights are globally reallocated, the Functional component will provide a total of 0.4. Accordingly, Functionality (m_1) has a global weight of 0.32 (0.4 x 0.8), and Product Evolution (m_2) has a global weight of 0.08 (0.4 x 0.2).

The individual judgments from each expert were entered into the AHP software and results from each expert were combined and calculated for the entire group. AHP can be applied easily with groups. Each member's assessments are evaluated for priorities and inconsistency using their own hierarchy, and then the group rollup is synthesized and calculated by taking the geometric mean of the final outcomes of the individual judgments. This approach provides an efficient way to build consensus since each expert can see where they stand and compare it to the group as a whole.

Results and Analysis

Evaluation Criteria

As discussed in Section 4.1, the experts comprised of five developers, five end users and five upper level managers. We calculated the overall priority for each of the criteria for each group and the result in terms of ranking is shown in Table 6.

In general, all experts from the three groups are very consistent with their choices, especially in choosing the most important variable, "Functionality." We can also see that "Licensing (m_3)", seems to be an important factor for the management team as it is ranked second. It makes sense for the management to have more concern about licensing because they are responsible for any legal issues in the company. The developers seem to value product evolution, support and community factors more than the other two groups as "Product Evolution (m_2) tied in the first place with Functionality (m_1). Support (m_8) and Community (m_5) are ranked second and third. Surprisingly, the developers did not think Code Quality (m_9) as an important variable influencing their OSS choice. As for the users, market penetration is highly influential in their choice of OSS.

Table 6. Survey Result

		Ranking		
	Criterion	Users	Management	Developers
Functional Features				
1	Functionality (m_1)	1	1	1
2	Product Evolution (m_2)	3	4	1
Non-functional Features				
3	Licensing (m_3)	5	2	4
4	Longevity/Pedigree (m_4)	7	8	8
5	Community (m_5)	6	6	3
6	Market Penetration (m_6)	2	3	5
7	Documentation Quality (m_7)	8	7	7
8	Support (m_8)	4	5	2
9	Code Quality (m_9)	9	8	6

We also calculated the overall priority for each of the criteria for the entire group and the result is shown in Table 7. The consistency ratio was calculated and a mean score of 0.062 was obtained, indicated an acceptable level of consistency. AHP has some tolerance for inconsistency, but Saaty recommends that a comparison with a consistency index below 0.1 is acceptable. If the group has a high inconsistency ratio (more than 0.1) segmenting might help to reveal where the differences in agreement are and why. By doing so, we can gain better understanding of the judgment process and consensus. In addition, we also calculated standard deviation to see how the scores are spread out.

From the result, we can see that the "Functional" criteria, which is the features the software has to cover, are considered to be more important for the experts compared to "Non-functional" features. We can also see that of the two Functional criteria, "Functionality (m_1) has the highest priority compared to "Product evolution (m_2)" or future improvements in functionality. A conclusion can be made that the experts recognized "Functionality" to have a greater value to the company and are integral in supporting its operation. The next aspect of importance is the future functionality that can be anticipated, that is, if there is already a plan of the future addition of the software.

We can also see that in terms of "Non-functional" features the importance of product's reputation in the markets and its value to the customers is also highly recognized as "Market Penetration (m_6)" is ranked third. It seems like the experts view the program worthwhile when there are many are other people using the software. It is interesting to see that "Licensing (m_3)" is ranked fourth; high on the list. Legal issues are definitely an important attribute to the experts in selecting the OSS application. "Support (m_8)" and "Community (m_8)" are ranked fifth and sixth. The experts might have viewed these two criteria as closely linked to each other because supports for most OSS application are available from both commercial and community. "Code Quality (m_9)", "Documentation (m_7)" and "Longevity/Pedigree (m_4)" are ranked seventh, eighth and ninth.

Table 7. Survey Result

	Criterion	Overall Priority	Overall Rank	Standard Deviations
Functional Features		**0.629**		**0.167**
1	Functionality (m_1)	0.662	1	0.150
2	Product Evolution (m_2)	0.285	2	0.150
Non-functional Features		**0.371**		**0.167**
3	Licensing (m_3)	0.213	4	0.081
4	Longevity/Pedigree (m_4)	0.066	9	0.032
5	Community (m_5)	0.119	6	0.058
6	Market Penetration (m_6)	0.238	3	0.077
7	Documentation Quality (m_7)	0.077	8	0.023
8	Support (m_8)	0.141	5	0.063
9	Code Quality (m_9)	0.080	7	0.020

OSS Application Selection

Figures 2a, 2b and 2c, showed the final selection of the OSS application made by the Users, Management and Developers respectively. The top two applications chosen by the users were Zimbra and OpenXchange; the management team chose OpenXchange followed closely by Zimbra while the developers chose OpenXchange and Scalix.

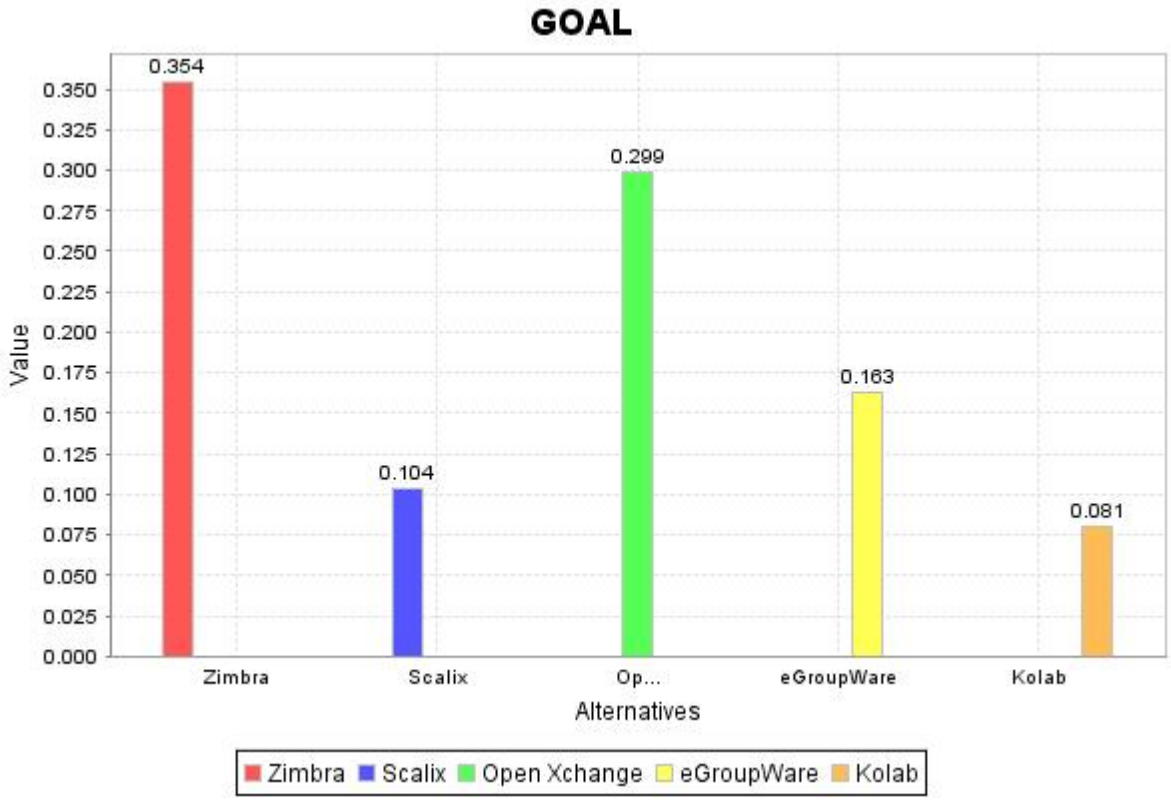

Figure 2a. Final OSS selection by the User

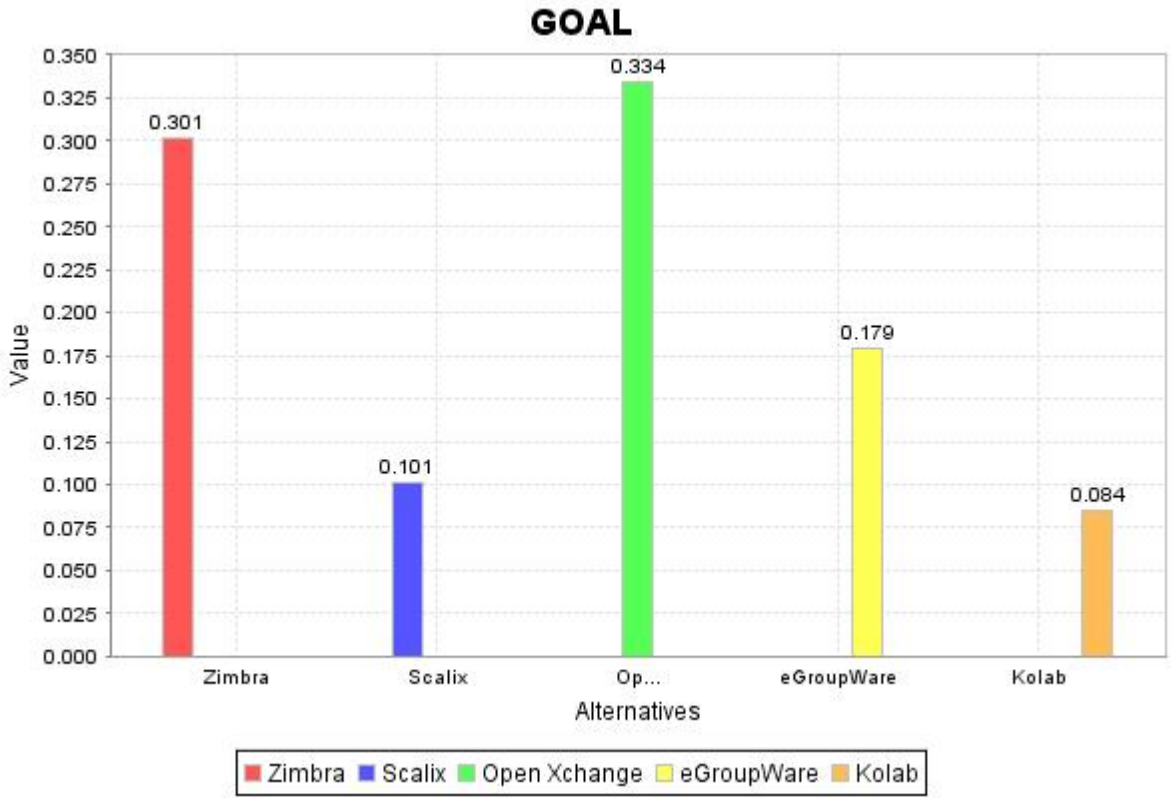

Figure 2b. Final OSS selection by the Management

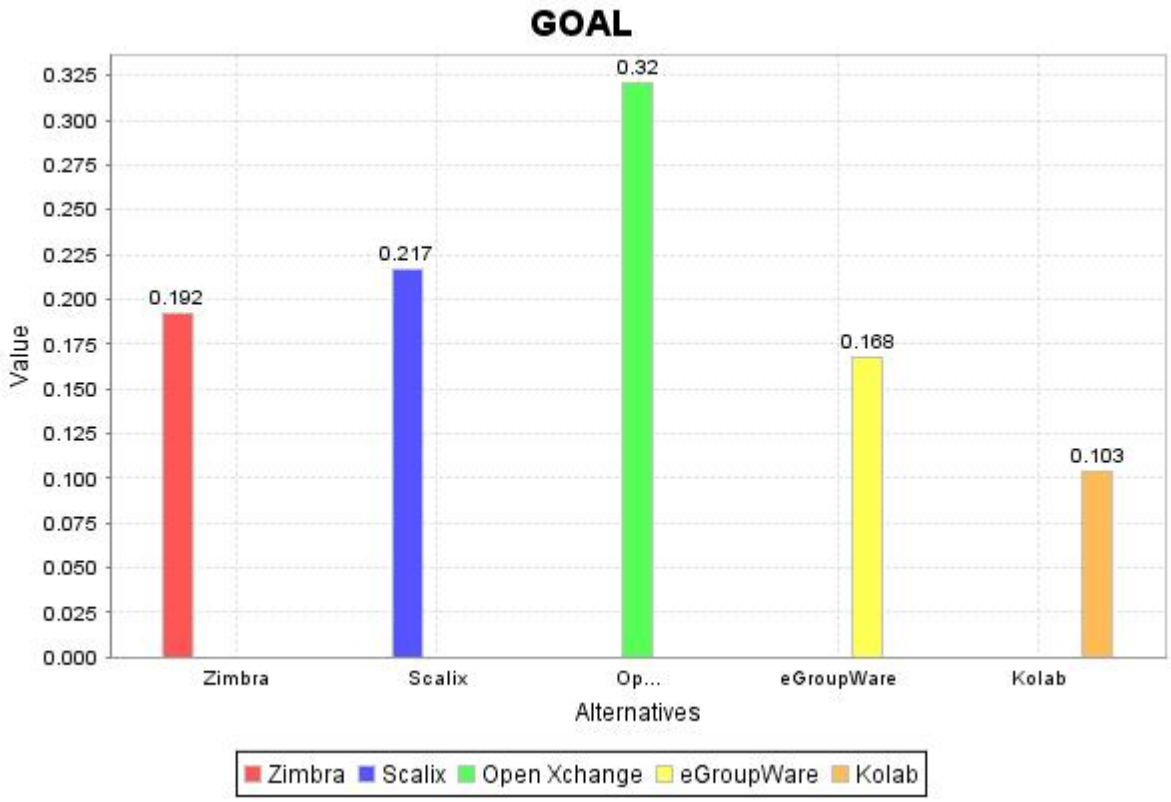

Figure 2c. Final OSS selection by the Developers

We can see that OpenXchange is the application of choice for both the developers and the management teams. Only the users' team chose Zimbra to be superior to the other applications. However, the users' team also viewed OpenXchange highly as it is ranked

second. Based on the result, we believe that the groups have achieved sufficient agreement and thus justified the use of geometric mean to average the group's preference without ignoring the differences of individual opinions.

The final results shown in Figure 3 shows that even though the experts have their own preferences in terms of the criteria (as discussed in section 6.1) overall they have almost reached a consensus in their selection of OSS application. The OSS application selected is OpenXchange, with an overall priority of 0.28. Zimbra came very close with an overall priority of 0.27.

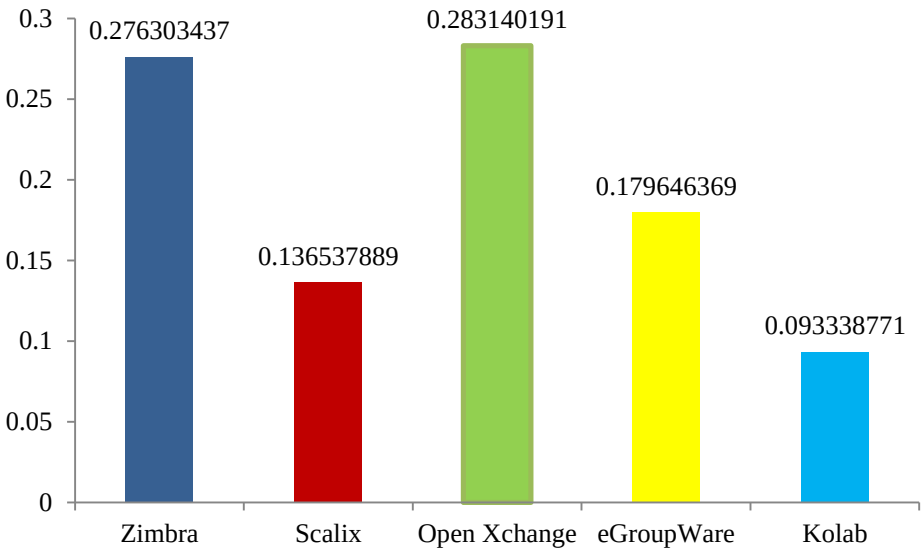

Figure 3. Final OSS selection by the group

Discussion and Implications

Selection and Evaluation of OSS criteria

As described in section 3, the selection of the evaluation criteria used in this study were constructed using the Open Source literature on the subject of evaluation of software and other existing models. As a result, we have identified nine important criteria: Functionality, Licensing, Project Velocity, Pedigree, Community, Market Penetration, Documentation Quality, Support, and Source Code.

The usual practice with regards to the selection of the evaluation criteria used at ITM was ad hoc and often informal where only the developers were involved in the selection process. According to the five developers who participated in this study, they usually made the selection based on their previous experience, Internet searches, recommendations and review from people in the developers' community, OSS community, vendors' recommendations and popularity of the software or the community itself. Often, when a developer has good knowledge or experience with a certain application, he omits any analysis and selects that application right away. But this approach is not systematic and can lead to all kinds of frustrations. As described by one of the users in our survey, "Sometimes, when we contact the IT support personnel for assistance, it turns out that they are not sure of how to use some

features in the software and in some cases they don't know if specific features are available in the software."

By using specific criteria and involving the end users in the evaluation process, as proposed in this study, an organization can increase the likelihood that what was selected today is still relevant tomorrow. In addition, by pre-defining the criteria prior to the evaluation process, users can gain familiarity and greater understanding of their needs. For example, when looking at Functional features (one of the criteria that we suggested in this study), the evaluators have to understand that the key goal of determining "functional requirements" is to capture the required behavior of a system in terms of functionality. If the OSS applications evaluated had some but all the functions needed, then evaluators should look at Product evolution or future improvements in functionality. The evaluators should understand what would it take to add those functions and whether it can be done internally or externally by others. If the developers are considering doing the improvements in house then they have to look at criteria such as code quality, documentation, support and community as well.

One of the main concerns that the end-users expressed during our first meeting was how to effectively perform the evaluations if they are not familiar with the criteria and the OSS applications that they are evaluating. We recommended the use of the internet to search for reviews and recommendations from social network such as OSS forums, and blogs, technology websites and magazines. By carefully going through each evaluation criteria, the users can educate themselves and effectively use their knowledge to test the requirements on the prototypes of each OSS applications.

Given the open nature of OSS communities, the end-users found it easy to get information on the OSS web site about documentation, license, release frequency, number of bugs, mailing lists, forums and other information. As described by one of the users, "I am surprised that I can actually get a lot done in relatively short time. I also learned a lot about the component and capabilities of each OSS application just by going to the OSS project websites."

The use of AHP-based evaluation model

The use of formal evaluation process such as the AHP-based evaluation model in selecting the OSS application was something that had never been done at ITM before. The developers found that the AHP-based evaluation model is very easy to use because it is very systematic and efficient. One of the developers said, "The process that we used before was chaotic and not repeatable, we usually go for the first application that fit our current requirements." Another developer added, "Now that we have a formal method we don't really have to rely solely on our experience and knowledge." Even though the proposed AHP-based evaluation model provides a selected set of criteria, it is flexible enough to adapt to different cases to suit a specific project or needs.

The end-users found this method to be especially helpful. Prior to this study, they were not involved in any decisions even though they are the ones who will be using the software. They are usually not aware of whether or not licenses are to be renewed or software are to be upgraded or changed. According to one of the users, they were left out in the evaluation process because most of them do not have technical background. However, in this study they found it very easy for them to get involved regardless of their lack of technical background. The AHP-based evaluation method used in this study is very systematic and easy to use.

The evaluators also found that in many cases, they have to reevaluate their judgments a few times in order to readjust the sensitivity of the selection process based on different applications. However, this process gave them better understanding of the impact of their priorities and the robustness of selection decisions to changes in those priorities. The AHP method enables decision makers to structure a decision making problem into a simple hierarchy, helping them to understand and simplify the problem. Most importantly the AHP-based evaluation model can be used in individual and group decision making, making it superior to other existing OSS evaluation models.

Limitations

As in any study, there are limitations to the present one. The validation of our approach employed only one IT company in the UAE. This company was small with limited application focus. Furthermore, we do not know to what extent cultural norms may have influenced the results such that a different result would have been obtained in another country. But these limitations can be addressed with future research.

Implications for research and practice

In order to address the limitations previously describe, the experiment conducted in this study will need to be repeated in various other companies, countries, application domains. Furthermore, it would be desirable to survey the ITM company after some period of time to see if their expectations have been met. It would also be desirable to survey other companies where similar open source application software has been selected using ad hoc techniques and compare their satisfaction level to that of ITM's, which used our decision making methodology.

We have demonstrated a practical approach to evaluating candidate open sources software by taking into account all stakeholder groups and using a rigorous decision making methodology. Over time organizations that use our approach should find that the decision making becomes easier as evaluators gain experience. Furthermore, our methodology can be adapted to include closed source software products by omitting criterion 9 (code quality) or by assuming some nominal value based on appropriate due diligence.

As such, our main contribution to existing research on OSS evaluation models is to help decision makers to better identify an appropriate OSS solution using a systematic and flexible approach where every member of the project team can be involved. We identified nine common evaluation criteria in selecting OSS solution however we understand that they might not be applicable in all cases - different project may have different requirements. In addition we also acknowledge that different decision makers have different expectations about OSS product. Therefore we proposed a model that is flexible and able to adapt to different cases. Existing models are more rigid and less flexible. Researchers and practitioners should therefore focus on developing a model that is sensitive to the situation and users need. A model like this would simplify the evaluation process and increase the actual adoption of OSS solution.

Conclusion

We introduced a rigorous decision making framework for selecting OSS using a set of criteria and AHP. We tested this methodology at an IT company in the UAE. Our main objective was to help decision makers to better identify an appropriate OSS solution using a systematic approach without the need for intensive performance testing. We believe that we have not only met our objective but at the same time produced a model that gives decision makers who are not familiar with OSS and its evaluation process insights into determining the best OSS application.

The results obtained in the case study were satisfactory, with experts from the three user groups making very consistent choices, especially with regard to the most important variable, "Functionality." Licensing seems to be an important factor for the management team while developers seem to value product evolution, support and community factors Surprisingly, the developers did not think Code Quality was an important variable influencing their OSS choice. As for the users, market penetration is highly influential in their choice of OSS.

Future experiments will be needed to revisit user satisfaction with respect to the OSS choice in the case study, and to further validate our decision making methodology across a wide variety of companies, application domains, and cultures.

References

Ahmad, Norita. Ph.D. Thesis. (2005) *The design, development and analysis of a multi criteria decision support system model: Performance Benchmarking of small to medium-sized manufacturing enterprise (SME)*. Rensselaer Polytechnic Institute.

Alan W. Brown , Kurt C. Wallnau. (1996). A Framework for Evaluating Software Technology, *IEEE Software*, 13(5), 39-49.

Bertoa M. F, J. M. Troya, and A. Vallecillo. (2003). A Survey on the Quality Information Provided by Software Component Vendors. In QAOOSE'03 Proceedings of the *7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, pp. 25–30.

Colombo E. and Francalanci C., (2004). Selecting CRM packages based on architectural, functional, and cost requirements: empirical validation of a hierarchical ranking model, *Requirements Engineering*, 186–203.

Confino, J. and Laplante, P. (2010). An Open Source Software Evaluation Model, *International Journal of Strategic Information Technology and Applications,* 1(1), 60-77.

Cristina Gacek , Budi Arief. (2004). The Many Meanings of Open Source, *IEEE Software*, 21(1), 34-40.

Crowston, K. and Howison, J., (2006). Hierarchy and centralization in free and open source software team communications, *Knowledge Technology & Policy,* 18(4), 65-85.

Crowston, K., Howison, J., and Annabi, H., (2006). Information systems success in free and open source software development: theory and measures, *Software Process—Improvement and Practice*, 11(2), 123-148.

Crowston, K., and Scozzi, B., (2002). Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development, *IEE Proceedings Software*, 149(1), 3-17.

Cruz D., T. Wieland, and A. Ziegler. 2006. Evaluation Criteria for Free/Open Source Software Products Based on Project Analysis. *Software Process: Improvement and Practice*, 11(2):107–122.

Deprez J.-C. and S. Alexandre. (2008). Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR and QSOS. In PROFES'2008 Proceedings of the 9th International Conference on Product-Focused Software Process Improvement, volume 5089/2008 of Lecture Notes in Computer Science, 189–203.

Donham. P. (2004) *Ten Rules for Evaluating Open Source Software. Point of view paper, Collaborative consulting*. Retrieved November 10, 2009, from http://www.collaborative.ws/leadership.php?subsection=27.

Duijnhouwer, F. W. and Widdows C. (2003) Capgemini Expert Letter Open Source Maturity Model, *Capgemini*. Retrieved November, 19, 2009, from http://www.seriouslyopen.org.

Elliott, M. and Scacchi, W. (2008), Mobilization of Software Developers: The Free Software Movement, *Information Technology & People*, 21(1), 4-33.

Feller, J., and Fitzgerald, B., (2002). *Understanding Open Source Software Development*, Addison-Wesley: NY.

Feller, J., Fitzgerald, B., Hissam, S. and Lakhani, K. (eds.), (2005). *Perspectives on Free and Open Source Software*, MIT Press, Cambridge: MA.

Gay, J. (ed.), (2002). *Free Software Free Society: Selected Essays of Richard M. Stallman*, GNU Press, Free Software Foundation, Boston: MA.

Golden, Bernard. (2004). *Succeeding With Open Source Software*, Addison-Wesley., Boston: MA.

Gorton I., A. Liu, and P. Brebner. (2003). Rigorous Evaluation of COTS Middleware Technology. *Computer*, 36(3):50–55.

Jean-Christophe Deprez, Simon Alexandre, (2009). *Comparing Assessment Methodologies for Free/Open Source Software: OpenBRR & QSOS*. Retrieved January 25, 2009, from http://www.qualoss.org/dissemination/DEPREZ_CompareFlOSSAssessMethodo-Camera-02.pdf.

Koch, S. (Ed.), (2005). *Free/Open Source Software Development*, Idea Group Publishing, Hershey: PA.

Lewis G. A. and E. J. Morris. (2004) From System Requirements to COTS Evaluation Criteria. In ICCBSS 2004 Proceedings of the *Third International Conference on COTS-Based Software Systems*, 2959/2004 of Lecture Notes in Computer Science, 159–168.

Li J., R. Conradi, C. Bunse, M. Torchiano, O. P. N. Slyngstad, and M. Morisio. (2009). Development with off-the-shelf components: 10 facts. *IEEE Software*, 26(2):2–9.

Majchrowski A. and J.-C. Deprez. (2008). An Operational Approach for Selecting Open Source Components in a Software Development Project. In EuroSPI'2008 Proceedings of the *15th European Conference on Software Process Improvement*, Volume 16 of Communications in Computer and Information Science, (pp. 176–188).

Martin, Robert C. (1999). *Designing Object Oriented Applications using UML*, 2d. ed., Robert C. Martin, Prentice Hall.

Merilinna J. and M. Matinlassi. (2006). State of the Art and Practice of OpenSource Component Integration. In EUROMICRO' 06 Proceedings of the *32nd EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE Computer Society* (pp. 170–177), Los Alamitos, CA, USA.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology*, 11(3), 309-346.

Norris J. S. (2004). Mission-critical Development with Open Source Software: Lessons Learned. IEEE Software, 21(1):42–49.

OpenBRR.org. (2009). Business Readiness Rating for Open Source©: *A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software, BRR, 2005*. Retrieved December 15, 2009, from http://www.openbrr.org.

OSMM Assessment, (2009). Retrieved Jan 25, 2009, from http://www.osspartner.com/portail/sections/accueil-public/evaluation-osmm.

OSS website. (2010). Retrieved May 12, 2010, from http://www.opensource.org/licenses.

QSOS. (2009). Retrieved December 15, 2009, from http://www.qsos.org/.

Raymond Eric S., (2001). *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*, O'Reilly & Associates, Inc., Sebastopol: CA.

Raymond, E.S. (1998). *The Cathedral and the Bazaar* [online]. Retrieved September 18, 2009, from: http://sagan.earthspace.net/~esr/writings/cathedral-bazaar/cathedral-bazaar.html.

Saaty T.L., (1980). *The Analytic Hierarchy Process*, McGraw- Hill:NY.

Torchiano M. and M. Morisio. (2004). Overlooked Aspects of COTS-Based Development. *IEEE Software*, 21(2), 88–93.

Wang H. and C. Wang. (2001). Open Source Software Adoption: A Status Report. *IEEE Software*, 18(2), 90–95.

Van Den Berg, K. (2005). *Finding open options: An Open Source software evaluation model with a case study on Course Management Systems*. Masters thesis, Tilburg University. Retrieved December 3, 2009, from http://www.karinvandenberg.nl/Thesis.pdf

Wikipedia, (2009). Open Source Software Assessment Methodologies, Retrieved December 15, 2009, from http://en.wikipedia.org/wiki/Open_source_software_assessment_methodologies.

Wheeler D. (2009). *How to Evaluate Open Source Software / Free Software (OSS/FS) Programs*, Retrieved November 15, 2009, from http://www.dwheeler.com/oss_fs_eval.html.